

Лекция 8.  
Производные классы.  
Часть 3.



НИЖЕГОРОДСКИЙ ИНСТИТУТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**НИИТ**

## Защищенные члены

```
class Employee {
    string name, surname;
protected:
    void retire();
    //...
};
```

```
void ret(Employee &re,
         Programmer &rp)
{
    re.retire();
    rp.retire();    // ошибка !!!
                   // ошибка !!!
}
```

```
class Programmer:
    public Employee {
    int team;
public:
    ~Programmer() { retire(); }
    //...
};
```

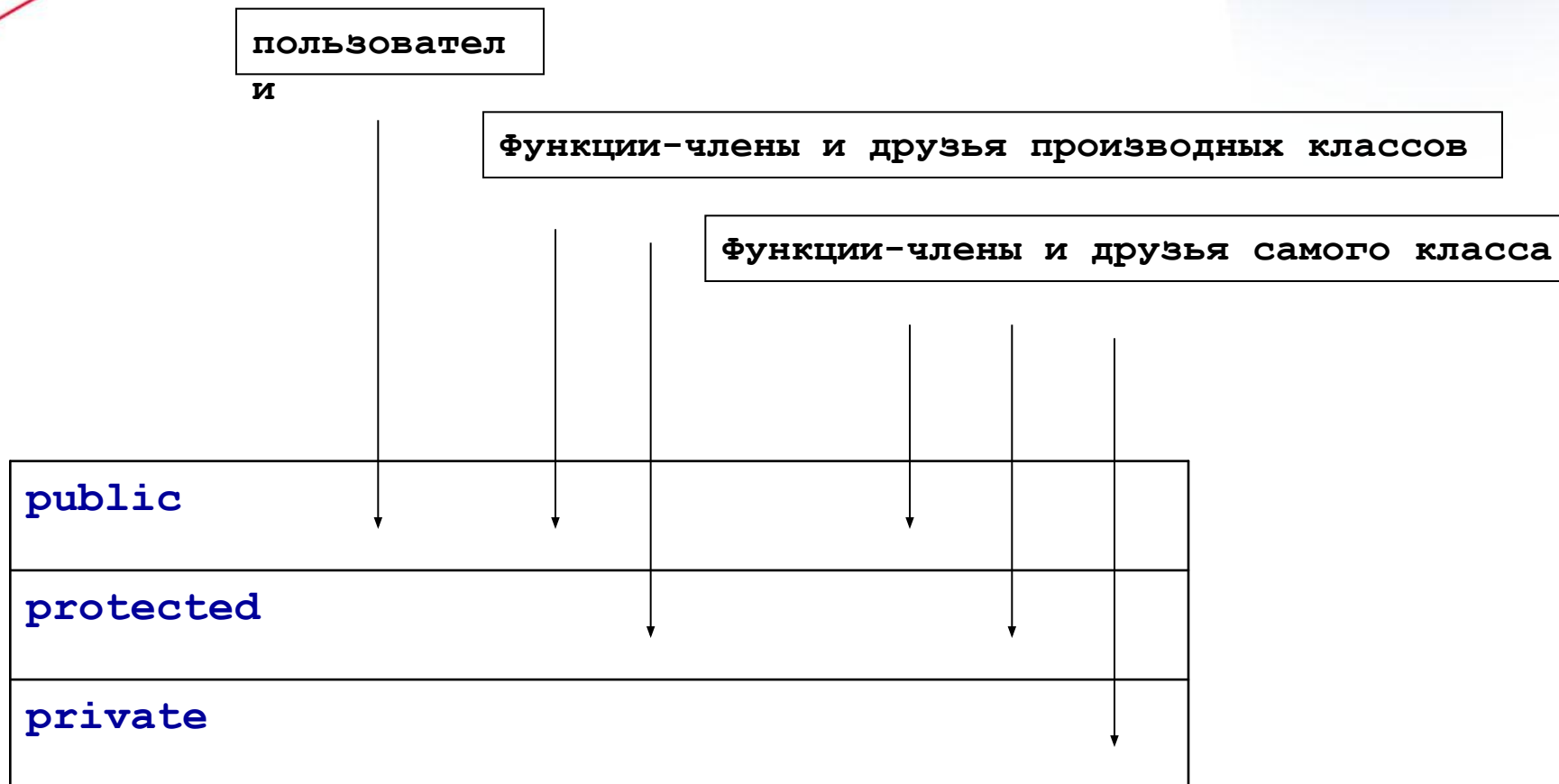
```
class Proj_Manager: public Employee
{ /*...*/};

class HR_Assistant: public Employee
{
    void fire(Proj_Manager *p)
    {
        p->retire();
        retire();    // ошибка !!!
    }
}
```

## Использование защищенных членов

- Производный класс имеет доступ к защищенным членам базового (но только для объектов собственного типа)
- Защищенные *данные* приводят к проблемам сопровождения
- Защищенные *функции* - хороший способ задания операций для использования в производных классах

# Управление доступом



## Доступ к базовым классам

- **public**
- **private**
- **protected**

## Правила доступа (public)

```
class Employee {
public:
    string name() const;
protected:
    void retire();
};

class Programmer:
    /*...*/ public Employee {
};

class Team_Leader:
    /*...*/ public Programmer{
};
```

```
void Team_Leader::~~Team_Leader()
{
    retire();
}
```

```
void f(Employee *emp,
        Programmer *prog,
        Team_Leader *tleader)
{
    prog->name();
    tleader->name();
    prog->retire(); // ошибка !!!
    emp = prog;
    prog = tleader;
    emp = tleader;
}
```

## Правила доступа (private)

```
class Stack {
public:
    void push(char);
    char pop();
protected:
    int max_size();
    int cur_size();
private:
    int max_size;
    //...
};
```

```
void f(Tough_Stack *p)
{
    p->put('a');
    p->pop();           // ошибка !!!
    Stack *pbase = p; // !!!
    pbase->pop();      // !!!
}
```

```
class Tough_Stack: private Stack {
public:
    void put(char c) { push(c); }
    char get() { if ( cur_size()>0 )
                 return pop(); }
};
```

```
class Semi_Tough_Stack :
    public Tough_Stack {
public:
    char pop(Tough_Stack *p) {
        Stack *pbase = p;           // !!!
        return pbase->pop();         // !!!
    }
};
```

## Правила доступа (protected)

```
class Unit {  
public:  
    bool move (int x, int y);  
    bool fire(int x, int y);  
    bool no_amm0();  
    bool reload();  
    void retreat();  
    void wound(int precent);  
private:  
    int X, Y;  
    int ammo;  
    int magazine;  
    int health;  
};
```

```
class Soldier: protected Unit  
{  
public:  
    bool move (int x, int y);  
    bool shoot(int x, int y);  
    void defend();  
protected:  
    void wound(int precent);  
};
```



## Правила доступа (protected)

```

bool Soldier::move(int x, int y)
{
    return Unit::move(x,y);
}

bool Soldier::shoot(int x, int y)
{
    if (no_ammo())
    { if (reload()==false)
        return false;
    }
    return fire(x,y);
}

void Soldier::wound(int percent)
{
    Unit::wound(percent);
    if (health()<20)
        retreat();
}

```

Nortel Networks Confidential

```

void madness()
{
    Soldier sol;
    if (!sol.shoot(x,y))
    {
        sol.retreat(); // ошибка !!!
    }

    if (sol.no_ammo()) // !!!
    {
        s1.wound(100); // ошибка !!!
    }
}

```

## Доступ к базовым классам

- Открытое наследование делает производный класс подтипом базового
- Защищенное и закрытое наследование используются для выражения деталей реализации
- Защищенные базовые классы полезны в иерархиях с дальнейшим наследованием
- Закрытые базовые классы полезны для “ужесточения интерфейса”

## Множественное наследование

```
class Storable_Process :
    public Process,
    public Storable {
    //...
};
```

```
void f(Storable_Process& rSP)
{
    rSP.read(); // Storable::read()
    rSP.run(); // Process::run()
    rSP.dump(std::cerr);
    rSP.stop(); // Process::stop()
    rSP.write(); // Storable::write()
}
```

```
void start(Process*);
bool check_filename(Storable*);

void susp(Storable_Process* pSP)
{
    if ( check_filename(pSP) )
    {
        start(pSP);
    }
}
```

## Множественное наследование (продолжение)

- Виртуальные функции работают как обычно

```
class Process {
    //...
    virtual void pending() = 0;
};
```

```
class Storable {
    //...
    virtual void prepare() = 0;
};
```

```
class Storable_Process :
    public Process,
    public Storable {
    //...
    virtual void pending();
    virtual void prepare();
};
```

## Разрешение неоднозначности

```
class Process {
    //...
    virtual debug_info* get_debug();
};
```

```
class Storable {
    //...
    virtual debug_info* get_debug();
};
```

```
void f(Storable_Process* pSP)
{
    debug_info* dip= pSP->get_debug();
    dip= pSP->Storable::get_debug();
    dip= pSP->Process::get_debug();
}
```

```
class Storable_Process :
    public Process,
    public Storable {
    //...
    virtual debug_info* get_debug()
    {
        debug_info* d1 =
            Storable::get_debug();
        debug_info* d2 =
            Process::get_debug();
        return d1->merge(d2);
    }
};
```

## Использование множественного наследования

- «Склеивание» двух не связанных классов вместе в качестве реализации третьего класса
- Реализация абстрактных классов

## Пример реализации абстрактного класса

```
class Drink {
public:
    virtual Liquid* drink() =0;
};
```

```
class Bottle {
public:
    void fill(Liquid*);
    Liquid* pour(Volume);
    void open();
    void break();
    bool opened() const;
    bool empty() const;
};
```

```
class BBeer: public Drink,
             protected Bottle {
/*...*/};
```

```
Liquid* BBeer::drink()
{
    if (!opened()) open();
    if (!empty())
        return pour(VOL_GULP);
    return NULL;
}
```

```
void get_drunk(BBeer* beer,
               Human *man)
{
    beer->break(); // ошибка !!!
    man->consume(beer->drink());
    Bottle *bottle = &beer; // !!!
    bottle->break(); // ошибка !!!
}
```

## Пример реализации абстрактного класса

```
bool Human::get_drunk(Drink* alc[], int num)
{
    for(int i=0; i<num || i_am_drunk() ; ++i)
    {
        Liquid *p = 0;
        while( (p=alc[i]->drink()) !=0 )
        {
            consume(p);
        }
    }
    return i_am_drunk();
}
```



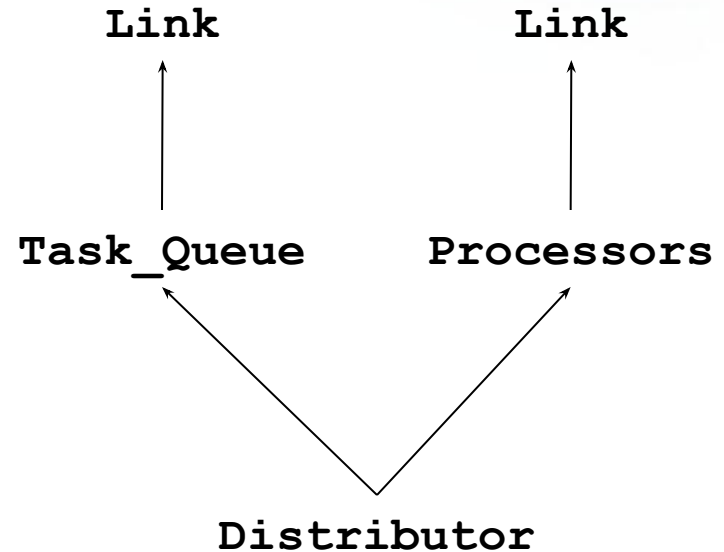
## Повторяющиеся базовые классы

```
class Link {
    //...
    Link* next();
};
```

```
class Task_Queue: public Link {
    //...
};
```

```
class Processors: public Link {
    //...
};
```

```
class Distributor : public Task_Queue,
                    public Processors {
    //...
};
```



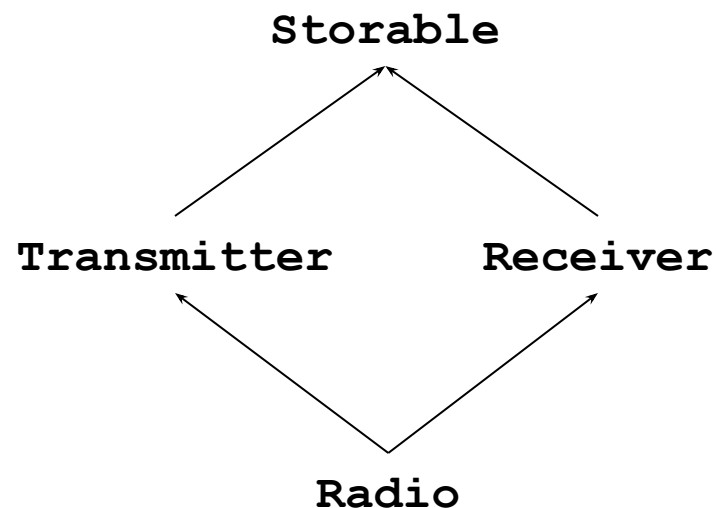
## Виртуальные базовые классы

```
class Storable {
public:
    Storable(const char*)
    virtual void read() =0;
    virtual void write() =0;
    virtual ~Storable() {write();}
private:
    const char* store;
};
```

```
class Radio : public Transmitter,
              public Receiver {
    Radio() : Storable("radio.stor")
    {}
    virtual void write()
    { Transmitter::write();
      Receiver::write(); }
};
```

```
class Transmitter: public virtual
Storable {
    //...
    virtual void write();
};
```

```
class Receiver: public virtual
Storable {
    //...
    virtual void write();
};
```



**Конец**